ECE385

Fall 2021

Final Project Report

Doodle Jump

Ge Yuhao, Lou Haina
D231, Jan.6, 2021
TA: Chenhao Wang

Zhejiang University-University of Illinois at Urbana-Champaign Institute
ZJU-UIUC

1 Overview

In this project, we build the game "Doodle Jump" on the DE2-115 board. Doodle_Jump (realised in 2009) is a popular game. In Doodle Jump, the aim is to guide a four-legged creature called "The Doodler" up a never-ending series of platforms without falling. The left side of the playing field is connected with the right side. Players use the key "A" and "D" to move the Doodler in the desired direction. Players can get a short boost from springs shown on some platforms. There are also monsters that the Doodler must avoid, shoot, or jump on to eliminate. The bullet aiming is performed by three keys "$\leftarrow \uparrow \rightarrow$". There is no definitive end to the game, but the end for each game play session happens when the player falls to the bottom of the screen or jumps into a monster. In our game, when the game starts, doodler will see a game start page and can press enter to turn to next playing image and control doodler jump up stairs to get higher scores. If doodler dies from dropping, it will free fall for a second and the screen will show the game over page with "Your score" shown. Then, if doodler wants to play again, he/she can press "Enter" again to turn to start game page to restart.

2 List of Features

Background The background of this game is a simple image of size 320 * 480 pixels. The left side of the playing field is connected with the right side, so the doodler can go through the left edge to the right edge and vise versa.

Start Frame A colorful and large 300 * 480 image display, player can press "Enter" to turn to next frame to play the game.

Figure 1: Start Frame

End Frame A colorful and large 300 * 480 display, it will show after doodle drops or jumps into the monster. Player can press "Enter" to restart to the start frame.

Figure 2: End Frame

Doodler Represents the player. It can jump between different platforms, which is controlled by the keys "A" and "D" to move left or right. The motion on the y-axis is affected by the gravity and the motion on the x-axis is constant speed with positive or negative sign. The Doodler's appearance will change as it goes left or right or is shooting bullets.

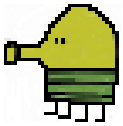


Figure 3: Go left

Figure 4: Shoot bullets

Figure 5: Go right

Platforms Many small platforms will be shown on the screen. When the doodler steps on to one of the platforms, it can jump up. There are 14 platforms at beginning. As there is no definitive end to the game, the new platforms will be generated while the doodler goes up and the positions are randomly generated. There will also be some platforms that can move left and right while the game gets harder.

Figure 6: Platform

Tools It will be generated randomly on some platforms. Doodler can get a short boost from springs when touching them.

Figure 7: Spring

Monster It will be generated randomly through the gaming process, and it can move in restricted area. Once it touches the doodler, the doodler will die. It can be defeated by the bullets and it will

also die when the doodler jumps on it.

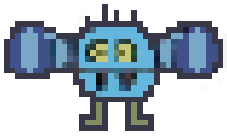


Figure 8: Monster

Bullets The doodler can shoot bullets to beat the monsters. The bullet aiming is performed by three keys "← ↑ →" to shoot towards up, upper left, or upper right.

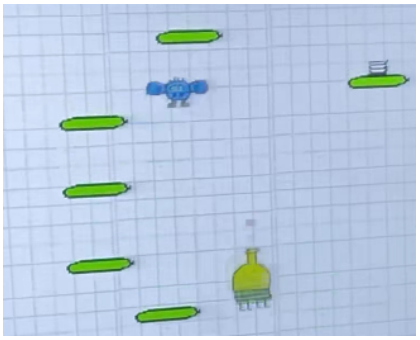


Figure 9: shoot

Score The score will show on the upper left of the screen to record the distance the doodler has travelled. It will also show on the end of the game to represent how well the player did.

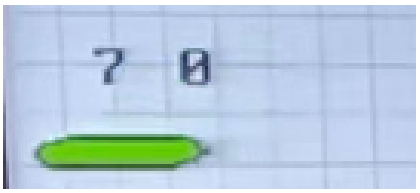


Figure 10: Score

Sound There is a background music produced by speakers for the game which will keep looping.

Difficulty The game will get harder when the doodler keeps going up. The difficulties are about the following aspects.

- There will be fewer platforms and the gap between different platforms will be larger.
- More platforms will begin to move left and right.
- The frequency of the appearance of monster will be larger.

3 Block Diagram

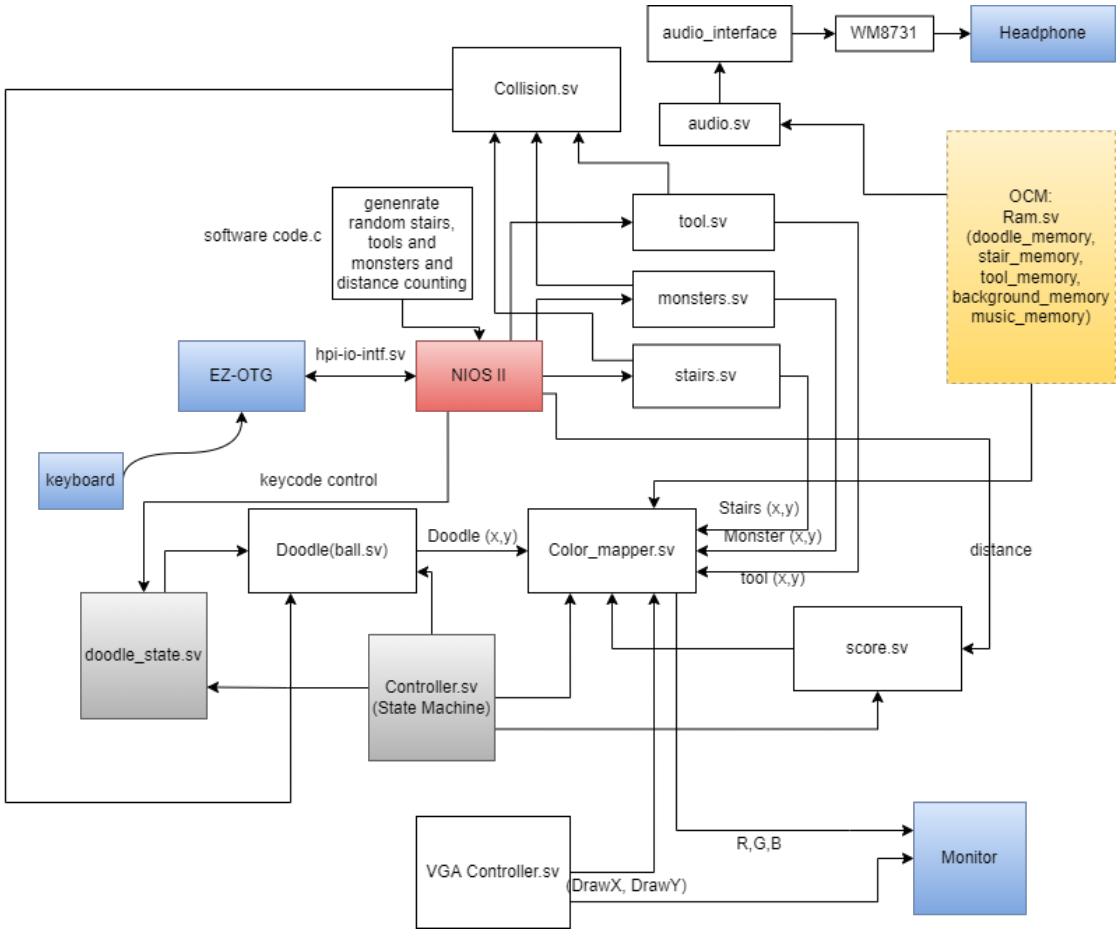


Figure 11: Block diagram

4 Description of the general flow of the circuit

The design is a mainly hardware based circuit, and we only use C code in software to generate random numbers and add some I/O in NIOS II to help hardware to read the numbers from keyboard and control the difficulty of the game. We use keyboard as our input device, while Monitor and headphone are output devices. EZ-OTG and WM8731 are not on FPGA but on DE2-115 and we use interface to communicate with them. With player input using keyboard, NIOS-II will transfer message to game logic modules which will output different signals to color_mapper module. With VGA controller that will tell color_mapper which pixel it is drawing, color_mapper will output corresponding RGB value and correct outputs(motion of sprites, start and end of game) will show on the monitor.

5 Module Description

5.1 Quartus modules

Module: score
Inputs: Clk, Reset, show, DrawX, DrawY, score_num
Output: is_score
Description: This module gives signal to the color mapper to indicate if the corrent pixel is the place

to display scores.
Purpose: Recode the total distance of the doodler, and make it show on the screen.

Module: digit_font
Inputs: addr
Output: data
Description: This module serves as the ROM, which defines the shape of the display number in pixel level.
Purpose: Serves as the sub-module which helps to show the digit on the screen.

Module: music
Inputs: Clk, Add
Output: music_content
Description: This module load the data from TXT file into the on chip memory of FPGA.
Purpose: This module helps to control the sound play of the board.

Module: audio
Inputs: Clk, Reset, INIT_FINISH, data_over, music_frequency
Output: INIT, Add
Description: This module communicates with the audio_interface.vhd, and uses state machine to control the frequency and length of the sound.
Purpose: This module helps to control the sound play of the board.

Module: controller
Inputs: Clk, Reset, frame_clk, keycode, death, drop
Output: show, restart
Description: This module is the biggest finite state machine for whole game, with five states: Start, Play, Dead, Drop and Halt. **Purpose:** it can control the whole game logic like start or end of game. With output signal show, start page of the game will show after pressing enter after death, and end page of the game will show after it hit the monster or drop.
Module: doodlestate
Inputa: Clk, frame_clk, Reset, keycode
Outputs: state
Description: This module controls the three states of doodler with different key codes pressed. Shoot-upwards for 1s after pressing direction keycode, toward left after pressing "A", toward right after pressing "D". **Purpose:** control the state of doodler to make it more flexible.
Module: ram
Inputa: Clk, read_address
Outputs: data_Out
Description: on chip memory, read txt files into ram. The content will lose after restart fpga and memory is very small compared to sram. **Purpose:** We use this module to store the index of our palette for many different components.
Module: Ball
Inputs: Clk, Reset, frame_clk, keycode, DrawX, DrawY
Output: is_ball
Description: This module creates the doodler instance and use SystemVerlog logic to control the moving and boucing of the doodler.
Purpose: Control the motion of the doodler.

Module: collision
Inputs: Clk, Reset, frame_clk, Ball_X_Pos, Ball_Y_Pos, Ball_Size, Ball_Y_Step, stair_x, stair_y, stair_size
Output: collision
Description: This module is used to detect if a doodler touch a stair
Purpose: This module gives signal to the ball.sv to realise the jumping function

Module: tot_distance
Inputs: Clk, Reset, drop, distance
Output: tot_distance
Description: This module sums each move of the doodler and give a total distance out.
Purpose: This module record the total distance of the doodler.

Module: bullets
Inputs: Clk, Reset, frame_clk, shoot, hit, direction, DrawX, DrawY, random_num, distance, Ball_X_Pos_out, Ball_Y_Pos_out, Ball_Size_out
Output: bullet_x, bullet_y, bullet_size, fly, is_bullet
Description: This module accept the keyboard inputs and shoots the bullets.
Purpose: This module is used to shoot bullets.

Module: bullets
Inputs: Clk, Reset, frame_clk, DrawX, DrawY, Ball_X_Pos_out, Ball_Y_Pos_out, Ball_Size_out,Ball_Y_Step_out, tool_x, tool_y, tool_size
Output: gain, is_tool
Description: This module will generate springs on some of the stairs accoding to the nio-II output signal.
Purpose: This module is used to generate tools which can be used by doodler.

Module: doo_mons
Inputs: Clk, Reset, frame_clk, Ball_X_Pos_out, Ball_Y_Pos_out, Ball_Size_out, Ball_Y_Step, stair_x,monster_x, monster_y, monster_size_x, monster_size_y, active
Output: death, beat_mons
Description: This module is used to detect if the doodler step on the monster or touch by monster
Purpose: This module give signal to the top level to denote if the monster is dead or the doodler is dead.

Module: stair
Inputs: Clk, Reset, DrawX, DrawY, random_num, distance,move_message, active_message, tool_message
Output: stair_x, stair_y, stair_size, is_stair, counter, tool_x, tool_y, tool_size
Description: This module is used to generate stairs. NIO-2 give messages to control the status of the stairs (Whether it has a spring, Whether it is moving), and the new position is randomly generated by NOI-2
Purpose: This module creates stairs.

Module: monsters
Inputs: Clk, Reset, DrawX, DrawY, random_num, distance, gene, hit, beat_mons
Output: monster_x, monster_y, monster_size_x,monster_size_y, active, is_monster
Description: This module is used to generate monsters. NIO-2 give messages to control whether there will be a monster on the next frame
Purpose: This module creates monsters.

Module: color_mapper
Inputs: DrawX, DrawY, is_ball
Output: VGA_R, VGA_G, VGA_B
Description: Decide which color to be output to VGA for each pixel
Purpose: This module can draw the picture of each frame.

Module: HexDriver
Inputs: Input
Output: Output

Description: The ASCII character value will be displayed onto the HexDriver of the DE2 board/
Purpose: The keyboard input will be displayed onto the HexDrive of the DE2 board, which helps us to know if the keyboard functions well.

Module: Lab8
Inputs: CLOCK_50, KEY, OTG_INT
Output: HEX0, HEX1, VGA_R, VGA_G, VGA_B, VGA_CLK, VGA_SYN_N, VGA_BLANK_N, VGA_VS, VGA_HS, OTG_DATA, OTG_ADDR, OTG_CS_N, OTG_RD_N, OTG_WR_N, OTG_RST_N, OTG_INT,DRAM_ADDR, DRAM_DQ, DRAM_BA, DRAM_DQM, DRAM_RAS_N, DRAM_CAS_N, DRAM_CKE, DRAM_WE_N, DRAM_CS_N, DRAM_CLK, AUD_ADCDAT, AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK, AUD_DACDAT, AUD_XCK, I2C_SCLK, I2C_SDAT
Description: This is the top level module of the final project.
Purpose: This module connect all the other modules, control the port communication of different modules, and control the communication between FPGA and USB interface and audio interface.

Module: VGA_controller
Inputs: Clk, Reset, VGA_CLK
Output: VGA_HS, VGA_VS, VGA_BLANK_N, VGA_SYNC_N, DrawX, DrawY
Description: Takes the data of the ball's current position on screen and updates it to the VGA display. Use the vertical and horizontal syncs signal to control the drawing of the VGA display.
Purpose: This module set the configuration of the screen such as the size, edge, and sync signals. It also records the present pixel we are drawing which enables the displaying function.

Module: hpi_io_intf
Inputs: Clk, Reset, from_sw_address, from_sw_data_out, from_sw_r, from_sw_w, from_sw_cs, from_sw_reset
Output: from_sw_data_in, OTG_ADDR, OTG_RD_N, OTG_WR_N, OTG_CS_N, OTG_RST_N
inout: OTG_DATA
Description: This module is the interface between NIOS II and EZ-OTG chip.
Purpose: This module sends the read, write, address, databuffer signals to the OTG chip to ensure correct data reads and writes. It also allows us to control the input output bus.

Module: vga_clk
Inputs: inclk0
Output: c0
Description: This module produce the clock signal which is used to control the VGA_controller.
Purpose: This module generates the 25MHz pixel clock for the VGA controller which allows the VGA controller to update the screen at clock to 60Hz.

5.2 Qsys Modules

5.2.1 Important modules

keycode This PIO allows the FPGA to accept signal from the keyboard.

tot_distance This PIO transfer the distance message from the hardware to the NIO-II to make the software know the current difficulty of the game.

monster This PIO allows the FPGA to accept signal from the keyboard to denote if its time to produce a monster on the top of the screen.

move_message This PIO transfers a series of bits to denote if a stair should be moving. Bit 0 means still stair and 1 means moving stair. The total length is 14 bits, and each bit corresponds to one stair.

active_message This PIO transfers a series of bits to denote if a stair should be moving. Bit 0 means a stair should be removed and 1 means normal stair. The total length is 14 bits, and each bit corresponds to one stair.

tool This PIO transfers a series of bits to denote if a stair should be moving. Bit 0 means a stair without spring and 1 means a stair with spring on it. The total length is 14 bits, and each bit corresponds to one stair.

5.2.2 Other modules

nios2_gen2_0: The nios2 block helps us to translate the C code into the SystemVerilog which can be excuted on the FPGA board.

Onchip_memory2_0 This is the memory block for the CPU which can store some data for the proigram.

Clk_0 Serves as the general clock for the entire system, which control all the other modules' clk signal. It can also reset the clk signal for other modules.

sdram As the space of Onchip_memory is limited, this block add another SDRAM to our system. And we have to use an SDRAM controller to interface.

sdram_pll This module generates the clock for the SDRAM. The pll add delay (3ns) to the SDRAM with delays to wait for the outputs to stabilize.

sysid_qsys_0 This block verifies the correct transfer of the software and hardware by looking back and forth between the C code and SystemVerilog to assure the data transfer is done in the correct format.

jtag_uart_0 Allows for terminal access for use in debugging the software. keycode: Reads data, updates the USB chip and then sends it to the software for logic instruction.

otg_hpi_address PIO that allows the desired address in memory of the SoC to be found that is sent from the software to the FPGA.

otg_hpi_data PIO that allows for cross transfer of data from the FPGA to software and the other way around. This PIO has a width of 16 bits which allows for sizable data transfer between the two.

otg_hpi_r PIO that allows the enable signal to read from the memory of the SoC.

otg_hpi_w PIO that allows the enable signal to write to the memory of the SoC.

otg_hpi_cs PIO that allows the enable signal turn on and off the memory of the SoC.

otg_hpi_reset PIO that allows for the reset signal of the memory of the SoC.

6 Design Procedure

6.1 Overview of the design procedure

The base of this project is the LAB8 "Bouncing Ball", which those circuits function well (VGA display, Keyboard input, ball's motion) the project building procedure is as follows.

First two weeks (before mid checkpoint)

- Basing on the ball's motion, add more complected features to the motion model. For example, add gravity to the model, enable the connection of the left boundary and the right boundary.
- Similar to the the procedure of creating a ball, create a stair and make it visible of the screen. Then add 13 more stairs with initialize position.
- Create a module to detect if the ball step onto one of the stairs to realise the bouncing function on the stair.
- To make sprites looks more attractive and more like the real game, we use ECE385 helper-tools provided by blackboard to draw pictures of doodle, stairs and make simple play background using on chip memory. We create different palette with 16 different colors per palette for each sprites and to save memory, we only load small piece of background picture, but repeat load it for our simple background.

Until the checkpoint, we realised a "game" that we can control a doodler jumping on some still stairs and the whole scene won't move.

Next two weeks (before demo)

- Try to make the scene to move down as the doodler jumps up. When the doodler is going to reach the upper half of the screen, make the doodler still and make all the stairs move down.
- When a stair is going to disappear on the button of the screen, reset the y-value and make it regenerated on the top of the screen.
- Add monster and bullet module. These modules are similar to the ball module.
- Modified the NIO-II software code. Add more PIOs to build connection between software and hardware. First, use random numbers produced by software the initialize the new position of each stair so the distribution of the stairs is irregular. Second, make the software periodically provide a signal telling the FPGA that a monster needs to be generated . Third, encode the status of each stair in three series of bits, use those bits produced by the NIO-II to control the hardware configurations. Then we can control the status of each stair through the software code.
- Add the distance module to calculate the total distance of the doodler, and this message is given to the NIO-II through PIO. Given this message, make the software being able to increase the game difficulty by automatically modified the above series of bits according to the distance signal. Then the game will getting harder as we will decrease the number of active stairs, increase the number of moving stairs, increase the master's showing frequency when distance getting larger.
- Add the collision checking modules between the monster and Doodler, the bullet and the monster. These modules decide the future of gaming.
- Apart from the motion of sprites, We also start to add state machine to control the whole game logic. We set five states: start, play, drop, dead and halt. We start game from start state which shows start page on the monitor, after player pressing "Enter", it will turn to play state and the game begins. If doodler misses its step, it will drop and turn to drop state, a nice falling animation starts for 50 frames and turn to dead state. The end game page will show on the monitor with score shown. If the doodler collide with monster, it will directly turn to dead state

without dropping. If we want to restart the game, player can press "Enter" to restart the game and go back to start page. Because we also use the habit of player to start game by pressing enter at start state, we add a halt state to make state stop at start state after dead. The state machine outputs signals to control whether it needs to restart and what image(start or end page) showing on the screen right now.

- After we make state machine and control signals, we begin to draw start and end image. Because we need too many pixels to draw and too many colors to store on memory, which also take too long to compile, we need to save memory for we only use on chip memory. I split the whole one page of image for four parts. For example, for 300x480 pixels picture, I store 4 300x120 pixels pictures in different memories by ram.sv, therefore, it will be loaded into OCM at the same time without using out of memory. And in order to keep reality of the image, I try to set different 16 colors of each palette for each four parts of the page, and load time is very quick and image is great of display.
- Add small state machine to control the state of doodler, turn left, turn right or shoot upwards. With different keycodes input, the upward state will last for one second while left-state and right-state doodler will last until the other keycode except 0 input.
- We also make score displaying on the monitor. By using Digit_ font.sv, we use distance recorded from software as score to generate address of digits we want and display at up left corner.
- Add the audio interface and the music module to play the background music.

7 State Diagram & Simulation Waveform

7.1 State machine

Many small state machines are used in different modules but are too trivial to included in the report. The most important state machine is used in our project to control the start and end of game logic. As shown in the state diagram below, we start game from start state which shows start page on the monitor, after player pressing "Enter", it will turn to play state and the game begins. If doodler misses its step, it will drop and turn to drop state, a nice falling animation starts for 50 frames and turn to dead state. The end game page will show on the monitor with score shown. If the doodler crushes on the monster, it will directly turn to dead state without dropping. If we want to restart the game, player can press "Enter" to restart the game and go back to start page. Because we also use the habit of player to start game by pressing enter at start state, we add a halt state to make state stop at start state after dead.

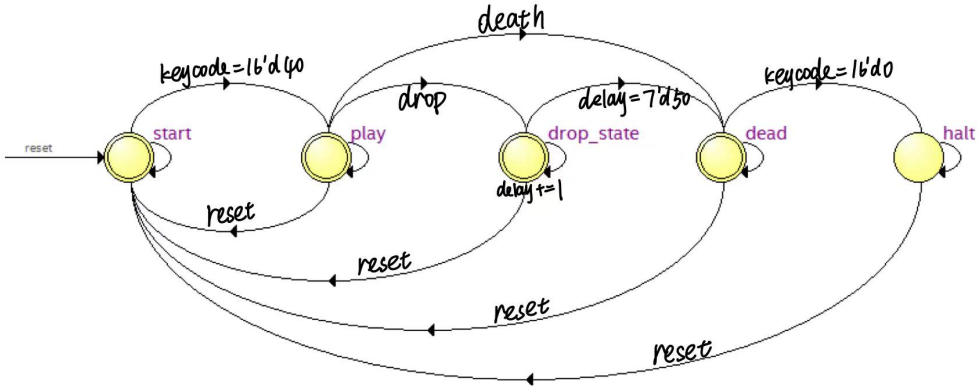
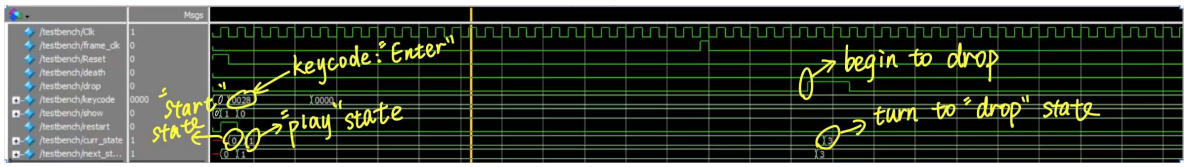


Figure 12: State diagram

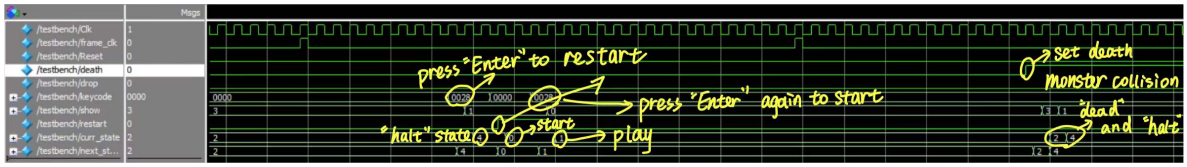
7.2 Simulation Waveforms

The Simulation Waveform for state machine controller is as follows.



(a) simulation1

(b) simulation2



(c) simulation3

Figure 13: Simulation for FSM

8 Design statistics

LUT	17845
DSP	0
Memory(BRAM)	3188736 bits
Flip-Flop	3086
Frequency	14.82MHz
Static Power	98.54mW
Dynamic Power	315.58mW
Total Power	532.34mW

Table 1: Design Resources and Statistics

9 Some known bugs

- When the doodler is exactly split by the left or right boundary, the x-axis motion will be stuck.
- Sometimes the spring will suddenly show up on a stair while the stair is at the middle of the screen, and sometimes the stairs will suddenly start moving while it is at the middle of the screen.
- When a still stair and a moving stair have same height, the picture of one stair will be wrong.
- The score displaying the total distance has a limited range of 0-99, when it reaches 100, the score showing will have some errors.

10 Accomplishments

Our project perfectly reproduces the "Doodle jump", the picture drawing is nice, the motion is smooth, and the props and process of the game are very close to the original game. Some outstanding and tricky designs are listed as follows.

- As we want to change the status of the stairs according to the game's level, the control logic must be complected so software might be a good choice. However, the status changing time of each component(stairs, monster, tools) of the game should be exact, a high efficiency is needed so hardware might be a good choice. Facing this, we use the idea of a combination of hardware and software through special data encoding method. We encode the status of each stair in three series of bits, each bit represents if a stair is moving, or if a stair is disappeared, or if there is a spring on the stair. With those bits produced by the NIO-II serving as message signal, we can control the hardware configurations through simple "if" statements. The procedure of deciding when those bits are going to change is handled in the C code in NIO-II. Then we can control the status of each stair through the software code.

```
*move_stairs =   0b01000100000000;
*active_stairs = 0b01011101010101;
*tool_message  = 0b00000001000100;
```

- Because we want to increase the speed of memory read, we only use on chip memory. However, on chip memory only has 2MB space to store image data and many other groups also encounter the problem of too long memory load time. Therefore, I try to split the large image into 4 pieces, because there are too many colors, in order to restore the game to a greater extent, I use four different palettes to draw the four pieces of image, which greatly increase the compilation speed as well as it has greater and more colorful look. We only have to compile for 4 minutes without loading music.

-

11 Conclusion

Through the unremitting efforts and cooperation of two team members, the final result of the project is very good which gained the "Best Design Award" of the course. Next two videos show the General Gaming Procedure and the Detailed Illustration of Each Feature. The whole project code is on the GitHub.